

Lifted Unit Propagation for Effective Grounding

Pashootan Vaezipoor¹, David Mitchell¹, and Maarten Mariën^{*2}

¹ Department of Computing Science, Simon Fraser University, Canada {pva6,mitchell}@cs.sfu.ca

² Department of Computer Science, Katholieke Universiteit Leuven, Belgium maartenm@cs.kuleuven.be

Abstract. A grounding of a formula ϕ over a given finite domain is a ground formula which is equivalent to ϕ on that domain. Very effective propositional solvers have made grounding-based methods for problem solving increasingly important, however for realistic problem domains and instances, the size of groundings is often problematic. A key technique in ground (e.g., SAT) solvers is unit propagation, which often significantly reduces ground formula size even before search begins. We define a “lifted” version of unit propagation which may be carried out prior to grounding, and describe integration of the resulting technique into grounding algorithms. We describe an implementation of the method in a bottom-up grounder, and an experimental study of its performance.

1 Introduction

Grounding is central in many systems for solving combinatorial problems based on declarative specifications. In grounding-based systems, a “grounder” combines a problem specification with a problem instance to produce a ground formula which represents the solutions for the instance. A solution (if there is one) is obtained by sending this formula to a “ground solver”, such as a SAT solver or propositional answer set programming (ASP) solver. Many systems have specifications given in extensions or restrictions of classical first order logic (FO), including: IDP [WMD08c], MXG [Moh04], Enfragmo [ATÜ⁺10,AWTM11], ASPPS [ET06], and Kodkod [TJ07]. Specifications for ASP systems, such as DLV [LPF⁺06] and clingo [GKK⁺08], are (extended) normal logic programs under stable model semantics.

Here our focus is grounding specifications in the form of FO formulas. In this setting, formula ϕ constitutes a specification of a problem (e.g., graph 3-colouring), and a problem instance is a finite structure \mathcal{A} (e.g., a graph). The grounder, roughly, must produce a ground formula ψ which is logically equivalent to ϕ over the domain of \mathcal{A} . Then ψ can be transformed into a propositional CNF formula, and given as input to a SAT solver. If a satisfying assignment is found, a solution to \mathcal{A} can be constructed from it. ASP systems use an analogous process.

A “naive” grounding of ϕ over a finite domain A can be obtained by replacing each sub-formula of the form $\exists x \psi(x)$ with $\bigvee_{a \in A} \psi(\tilde{a})$, where \tilde{a} is a constant symbol which denotes domain element a , and similarly replacing each subformula $\forall x \psi(x)$ with a conjunction. For a fixed FO formula ϕ , this can be done in time polynomial in $|A|$. Most grounders use refinements of this method, implemented top-down or bottom-up, and perform well on simple benchmark problems and small instances. However, as we tackle more realistic problems with complex specifications and instances having large domains, the groundings produced can become prohibitively large. This can be the case even when the formulas are “not too hard”. That is, the system performance is poor because of time spent generating and manipulating this large ground formula, yet an essentially equivalent but smaller formula can be solved in reasonable time. This work represents one direction in our group’s efforts to develop techniques which scale effectively to complex specifications and large instances.

Most SAT solvers begin by executing unit propagation (UP) on the input formula (perhaps with other “pre-processing”). This initial application of UP often eliminates a large number of variables and clauses, and is done very fast. However, it may be too late: the system has already spent a good deal of time generating large but rather uninteresting (parts of) ground formulas, transforming them to CNF, moving them from the grounder to the SAT solver, building the SAT solver’s data structures, etc. This suggests trying to execute a process similar to UP *before or during grounding*.

One version of this idea was introduced in [WMD08b,WMD10]. The method presented there involves computing a *symbolic* and *incomplete* representation of the information that UP could derive, obtained

* This author’s contributions to this paper were made while he was a post-doctoral fellow at SFU.

from ϕ alone without reference to a particular instance structure. For brevity, we refer to that method as **GWB**, for “Grounding with Bounds”. In [WMD08b,WMD10], the top-down grounder **GIDL** [WMD08a] is modified to use this information, and experiments indicate it significantly reduces the size of groundings without taking unreasonable time.

An alternate approach is to construct a *concrete* and *complete* representation of the information that UP can derive about a grounding of ϕ over \mathcal{A} , and use this information during grounding to reduce grounding size. This paper presents such a method, which we call *lifted unit propagation* (**LUP**). (The authors of the **GWB** papers considered this approach also [DW08], but to our knowledge did not implement it or report on it. The relationship between **GWB** and **LUP** is discussed further in Section 7.) The **LUP** method is roughly as follows.

1. Modify instance structure \mathcal{A} to produce a new (partial) structure which contains information equivalent to that derived by executing UP on the CNF formula obtained from a grounding of ϕ over \mathcal{A} . We call this new partial structure the **LUP** structure for ϕ and \mathcal{A} , denoted $\mathcal{LUP}(\phi, \mathcal{A})$.
2. Run a modified (top-down or bottom-up) grounding algorithm which takes as input, ϕ and $\mathcal{LUP}(\phi, \mathcal{A})$, and produces a grounding of ϕ over \mathcal{A} .

The modification in step 2 relies on the idea that a tuple in $\mathcal{LUP}(\phi, \mathcal{A})$ indicates that a particular sub-formula has the same (known) truth value in every model. Thus, that subformula may be replaced with its truth value. The CNF formula obtained by grounding over $\mathcal{LUP}(\phi, \mathcal{A})$ is at most as large as the formula that results from producing the naive grounding and then executing UP on it. Sometimes it is much smaller than this, because the grounding method naturally eliminates some autark sub-formulas which UP does not eliminate, as explained in Sections 3 and 6.

We compute the **LUP** structure by constructing, from ϕ , an inductive definition of the relations of the **LUP** structure for ϕ and \mathcal{A} (see Section 4). We implemented a semi-naive method for evaluating this inductive definition, based on relational algebra, within our grounder **Enfragmo**. (We also computed these definitions using the ASP grounders **gringo** and **DLV**, but these were not faster.)

For top-down grounding (see Section 3), we modify the naive recursive algorithm to check the derived information in $\mathcal{LUP}(\phi, \mathcal{A})$ at the time of instantiating each sub-formula of ϕ . This algorithm is presented primarily for expository purposes, and is similar to the modified top-down algorithm used for **GWB** in **GIDL**.

For bottom-up grounding (see Section 5), we revise the bottom-up grounding method based on extended relational algebra described in [MTHM06,PLTG07], which is the basis of grounders our group has been developing. The change required to ground using $\mathcal{LUP}(\phi, \mathcal{A})$ is a simple revision to the base case.

In Section 6 we present an experimental evaluation of the performance of our grounder **Enfragmo** with **LUP**. This evaluation is limited by the fact that our **LUP** implementation does not support specifications with arithmetic or aggregates, and a shortage of interesting benchmarks which have natural specifications without these features. Within the limited domains we have tested to date, we found:

1. CNF formulas produced by **Enfragmo** with **LUP** are always smaller than the result of running UP on the CNF formula produced by **Enfragmo** without **LUP**, and in some cases much smaller.
2. CNF formulas produced by **Enfragmo** with **LUP** are always smaller than the ground formulas produced by **GIDL**, with or without **GWB** turned on.
3. Grounding over $\mathcal{LUP}(\phi, \mathcal{A})$ is always slower than grounding without, but CNF transformation with **LUP** is almost always faster than without.
4. Total solving time for **Enfragmo** with **LUP** is sometimes significantly less than that of **Enfragmo** without **LUP**, but in other cases is somewhat greater.
5. **Enfragmo** with **LUP** and the SAT solver **MiniSat** always runs faster than the IDP system (**GIDL** with ground solver **MINISAT(ID)**), with or without the **GWB** method turned on in **GIDL**.

Determining the extent to which these observations generalize is future work.

2 FO Model Expansion and Grounding

A natural formalization of combinatorial search problems and their specifications is as the logical task of model expansion (MX) [MT11]. Here, we define MX for the special case of FO. Recall that a structure \mathcal{B}

for vocabulary $\sigma \cup \varepsilon$ is an expansion of σ -structure \mathcal{A} iff \mathcal{A} and \mathcal{B} have the same domain ($A = B$), and interpret their common vocabulary identically, i.e., for each symbol R of σ , $R^{\mathcal{B}} = R^{\mathcal{A}}$. Also, if \mathcal{B} is an expansion of σ -structure \mathcal{A} , then \mathcal{A} is the reduct of \mathcal{B} defined by σ .

Definition 1 (Model Expansion for FO).

Given: A FO formula ϕ on vocabulary $\sigma \cup \varepsilon$ and a σ -structure \mathcal{A} ,

Find: an expansion \mathcal{B} of \mathcal{A} that satisfies ϕ .

In the present context, the formula ϕ constitutes a problem specification, the structure \mathcal{A} a problem instance, and expansions of \mathcal{A} which satisfy ϕ are solutions for \mathcal{A} . Thus, we call the vocabulary of \mathcal{A} , the *instance* vocabulary, denoted by σ , and ε the *expansion* vocabulary. We sometimes say ϕ is \mathcal{A} -satisfiable if there exists an expansion \mathcal{B} of \mathcal{A} that satisfies ϕ .

Example 1. Consider the following formula ϕ :

$$\begin{aligned} \forall x[(R(x) \vee B(x) \vee G(x)) \wedge \neg(R(x) \wedge B(x)) \wedge \neg(R(x) \wedge G(x)) \wedge \neg(B(x) \wedge G(x))] \\ \wedge \forall x \forall y[E(x, y) \supset (\neg(R(x) \wedge R(y)) \wedge \neg(B(x) \wedge B(y)) \wedge \neg(G(x) \wedge G(y)))]. \end{aligned}$$

A finite structure \mathcal{A} over vocabulary $\sigma = \{E\}$, where E is a binary relation symbol, is a graph. Given graph $\mathcal{A} = \mathcal{G} = (V; E)$, there is an expansion \mathcal{B} of \mathcal{A} that satisfies ϕ , iff \mathcal{G} is 3-colourable. So ϕ constitutes a specification of the problem of graph 3-colouring. To illustrate:

$$\underbrace{(V; \overbrace{E^{\mathcal{A}}}^{\mathcal{A}}, R^{\mathcal{B}}, B^{\mathcal{B}}, G^{\mathcal{B}})}_{\mathcal{B}} \models \phi$$

An interpretation for the expansion vocabulary $\varepsilon := \{R, B, G\}$ given by structure \mathcal{B} is a colouring of \mathcal{G} , and the proper 3-colourings of \mathcal{G} are the interpretations of ε in structures \mathcal{B} that satisfy ϕ .

2.1 Grounding for Model Expansion

Given ϕ and \mathcal{A} , we want to produce a CNF formula (for input to a SAT solver), which represents the solutions to \mathcal{A} . We do this in two steps: grounding, followed by transformation to CNF. The grounding step produces a ground formula ψ which is equivalent to ϕ over expansions of \mathcal{A} . To produce ψ , we bring domain elements into the syntax by expanding the vocabulary with a new constant symbol for each domain element. For A , the domain of \mathcal{A} , we denote this set of constants by \tilde{A} . For each $a \in A$, we write \tilde{a} for the corresponding symbol in \tilde{A} . We also write \tilde{a} , where \tilde{a} is a tuple.

Definition 2 (Grounding of ϕ over \mathcal{A}). Let ϕ be a formula of vocabulary $\sigma \cup \varepsilon$, \mathcal{A} be a finite σ -structure, and ψ be a ground formula of vocabulary μ , where $\mu \supseteq \sigma \cup \varepsilon \cup \tilde{A}$. Then ψ is a grounding of ϕ over \mathcal{A} if and only if:

1. if ϕ is \mathcal{A} -satisfiable then ψ is \mathcal{A} -satisfiable;
2. if \mathcal{B} is a μ -structure which is an expansion of \mathcal{A} and gives \tilde{A} the intended interpretation, and $\mathcal{B} \models \psi$, then $\mathcal{B} \models \phi$.

We call ψ a *reduced grounding* if it contains no symbols of the instance vocabulary σ .

Definition 2 is a slight generalization of that used in [MTHM06, PLTG07], in that it allows ψ to have vocabulary symbols not in $\sigma \cup \varepsilon \cup \tilde{A}$. This generalization allows us to apply a Tseitin-style CNF transformation in such a way that the resulting CNF formula is still a grounding of ϕ over \mathcal{A} . If \mathcal{B} is an expansion of \mathcal{A} satisfying ψ , then the reduct of \mathcal{B} defined by $\sigma \cup \varepsilon$ is an expansion of \mathcal{A} that satisfies ϕ . For the remainder of the paper, we assume that ϕ is in negation normal form (NNF), i.e., negations are applied only to atoms. Any formula may be transformed in linear time to an equivalent formula in NNF.

Algorithm 1 produces the “naive grounding” of ϕ over \mathcal{A} mentioned in the introduction. A substitution is a set of pairs (x/a) , where x is a variable and a a constant symbol. If θ is a substitution, then $\phi[\theta]$ denotes

Algorithm 1 Top-Down Naive Grounding of NNF formula ϕ over \mathcal{A}

$$\text{NaiveGnd}_{\mathcal{A}}(\phi, \theta) = \begin{cases} P(\bar{x})[\theta] & \text{if } \phi \text{ is an atom } P(\bar{x}) \\ \neg P(\bar{x})[\theta] & \text{if } \phi \text{ is a negated atom } \neg P(\bar{x}) \\ \bigwedge_i \text{NaiveGnd}_{\mathcal{A}}(\psi_i, \theta) & \text{if } \phi = \bigwedge_i \psi_i \\ \bigvee_i \text{NaiveGnd}_{\mathcal{A}}(\psi_i, \theta) & \text{if } \phi = \bigvee_i \psi_i \\ \bigwedge_{a \in \mathcal{A}} \text{NaiveGnd}_{\mathcal{A}}(\psi, [\theta \cup (x/\tilde{a})]) & \text{if } \phi = \forall x \psi \\ \bigvee_{a \in \mathcal{A}} \text{NaiveGnd}_{\mathcal{A}}(\psi, [\theta \cup (x/\tilde{a})]) & \text{if } \phi = \exists x \psi \end{cases}$$

the result of substituting constant symbol a for each free occurrence of variable x in ϕ , for every (x/a) in θ . We allow conjunction and disjunction to be connectives of arbitrary arity. That is $(\bigwedge \phi_1 \phi_2 \phi_3)$ is a formula, not just an abbreviation for some parenthesization of $(\phi_1 \wedge \phi_2 \wedge \phi_3)$. The initial call to Algorithm 1 is $\text{NaiveGnd}_{\mathcal{A}}(\phi, \emptyset)$, where \emptyset is the empty substitution.

The ground formula produced by Algorithm 1 is not a grounding of ϕ over \mathcal{A} (according to Definition 2), because it does not take into account the interpretations of σ given by \mathcal{A} . To produce a grounding of ϕ over \mathcal{A} , we may conjoin a set of atoms giving that information. In the remainder of the paper, we write $\text{NaiveGnd}_{\mathcal{A}}(\phi)$ for the result of calling $\text{NaiveGnd}_{\mathcal{A}}(\phi, \emptyset)$ and conjoining ground atoms to it to produce a grounding of ϕ over \mathcal{A} . We may also produce a reduced grounding from $\text{NaiveGnd}_{\mathcal{A}}(\phi, \emptyset)$ by “evaluating out” all atoms of the instance vocabulary. The groundings produced by algorithms described later in this paper can be obtained by simplifying out certain sub-formulas of $\text{NaiveGnd}_{\mathcal{A}}(\phi)$.

2.2 Transformation to CNF and Unit Propagation

To transform a ground formula to CNF, we employ the method of Tseitin [Tse68] with two modifications. The method, usually presented for propositional formulas, involves adding a new atom corresponding to each sub-formula. Here, we use a version for ground FO formulas, so the resulting CNF formula is also a ground FO formula, over vocabulary $\tau = \sigma \cup \varepsilon \cup \tilde{A} \cup \omega$, where ω is a set of new relation symbols which we call “Tseitin symbols”. To be precise, ω consists of a new k -ary relation symbol $[\psi]$ for each subformula ψ of ϕ with k free variables. We also formulate the transformation for formulas in which conjunction and disjunction may have arbitrary arity.

Let $\gamma = \text{NaiveGnd}_{\mathcal{A}}(\phi, \emptyset)$. Each subformula α of γ is a grounding over \mathcal{A} of a substitution instance $\psi(\bar{x})[\theta]$, of some subformula ψ of ϕ with free variables \bar{x} . To describe the CNF transformation, it is useful to think of labelling the subformulas of γ during grounding as follows. If α is a grounding of formula $\psi(\bar{x})[\theta]$, label α with the ground atom $[\psi](\bar{x})[\theta]$. To minimize notation, we will denote this atom by $\hat{\alpha}$, setting $\hat{\alpha}$ to α if α is an atom. Now, we have for each sub-formula α of the ground formula ψ , a unique ground atom $\hat{\alpha}$, and we carry out the Tseitin transformation to CNF using these atoms.

Definition 3. For ground formula ψ , we denote by $\text{CNF}(\psi)$ the following set of ground clauses. For each sub-formula α of ψ of form $(\bigwedge_i \alpha_i)$, include in $\text{CNF}(\psi)$ the set of clauses $\{(\neg \hat{\alpha} \vee \hat{\alpha}_i)\} \cup \{(\bigvee_i \neg \hat{\alpha}_i \vee \hat{\alpha})\}$, and similarly for the other connectives.

If ψ is a grounding of ϕ over \mathcal{A} , then $\text{CNF}(\psi)$ is also. The models of ψ are exactly the reducts of the models of $\text{CNF}(\psi)$ defined by $\sigma \cup \varepsilon \cup \tilde{A}$. $\text{CNF}(\psi)$ can trivially be viewed as a propositional CNF formula. This propositional formula can be sent to a SAT solver, and if a satisfying assignment is found, a model of ϕ which is an expansion of \mathcal{A} can be constructed from it.

Definition 4 (UP(γ)). Let γ be a ground FO formula in CNF. Define $\text{UP}(\gamma)$, the result of applying unit propagation to γ , to be the fixed point of the following operation:

If γ contains a unit clause (l) , delete from each clause of γ every occurrence of $\neg l$, and delete from γ every clause containing l .

Now, $\text{CNF}(\text{NaiveGND}_{\mathcal{A}}(\phi))$ is the result of producing the naive grounding of ϕ over \mathcal{A} , and transforming it to CNF in the standard way, and $\text{UP}(\text{CNF}(\text{NaiveGND}_{\mathcal{A}}(\phi)))$ is the formula obtained after simplifying it by executing unit propagation. These two formulas provide reference points for measuring the reduction in ground formula size obtained by LUP.

3 Bound Structures and Top-down Grounding

We present grounding algorithms, in this section and in Section 4, which produce groundings of ϕ over a class of partial structures, which we call bound structures, related to \mathcal{A} . The structure $\mathcal{LUP}(\phi, \mathcal{A})$ is a particular bound structure. In this section, we define partial structures and bound structures, and then present a top-down grounding algorithm. The formalization of bound structures here, and of $\mathcal{LUP}(\phi, \mathcal{A})$ in Section 4, are ours, although a similar formalization was implicit in [DW08].

3.1 Partial Structures and Bound Structures

A relational τ -structure \mathcal{A} consists of a domain A together with a relation $R^{\mathcal{A}} \subset A^k$ for each k -ary relation symbol of τ . To talk about partial structures, in which the interpretation of a relation symbol may be only partially defined, it is convenient to view a structure in terms of the characteristic functions of the relations. Partial τ -structure \mathcal{A} consists of a domain A together with a k -ary function $\chi_R^{\mathcal{A}} : A^k \rightarrow \{\top, \perp, \infty\}$, for each k -ary relation symbol R of τ . Here, as elsewhere, \top denotes true, \perp denotes false, and ∞ denotes undefined. If each of these characteristic functions is total, then \mathcal{A} is total. We may sometimes abuse terminology and call a relation partial, meaning the characteristic function interpreting the relation symbol in question is partial.

Assume the natural adaptation of standard FO semantics to the case of partial relations, e.g. with Kleene's 3-valued semantics [Kle52]. For any (total) τ -structure \mathcal{B} , each τ -sentence ϕ is either true or false in \mathcal{B} ($\mathcal{B} \models \phi$ or $\mathcal{B} \not\models \phi$), and each τ -formula $\phi(\bar{x})$ with free variables \bar{x} , defines a relation

$$\phi^{\mathcal{B}} = \{\bar{a} \in A^{|\bar{x}|} : \mathcal{B} \models \phi(\bar{x})[\bar{x}/\bar{a}]\}. \quad (1)$$

Similarly, for any partial τ -structure, each τ -sentence is either true, false or undetermined in \mathcal{B} , and each τ -formula $\phi(\bar{x})$ with free variables \bar{x} defines a partial function

$$\chi_{\phi}^{\mathcal{A}} : A^k \rightarrow \{\top, \perp, \infty\}. \quad (2)$$

In the case $\chi_{\phi}^{\mathcal{A}}$ is total, it is the characteristic function of the relation (1).

There is a natural partial order on partial structures for any vocabulary τ , which we may denote by \leq , where $\mathcal{A} \leq \mathcal{B}$ iff \mathcal{A} and \mathcal{B} agree at all points where they are both defined, and \mathcal{B} is defined at every point \mathcal{A} is. If $\mathcal{A} \leq \mathcal{B}$, we may say that \mathcal{B} is a strengthening of \mathcal{A} . When convenient, if the vocabulary of \mathcal{A} is a proper subset of that of \mathcal{B} , we may still call \mathcal{B} a strengthening of \mathcal{A} , taking \mathcal{A} to leave all symbols not in its vocabulary, completely undefined. We will call \mathcal{B} a conservative strengthening of \mathcal{A} with respect to formula ϕ if \mathcal{B} is a strengthening of \mathcal{A} and in addition every total structure which is a strengthening of \mathcal{A} and a model of ϕ is also a strengthening of \mathcal{B} . (Intuitively, we could ground ϕ over \mathcal{B} instead of \mathcal{A} , and not lose any intended models.)

The specific structures of interest are over a vocabulary expanding the vocabulary of ϕ in a certain way. We will call a vocabulary τ a Tseitin vocabulary for ϕ if it contains, in addition to the symbols of ϕ , the set ω of Tseitin symbols for ϕ . We call a τ -structure a “Tseitin structure for ϕ ” if the interpretations of the Tseitin symbols respect the special role of those symbols in the Tseitin transformation. For example, if α is $\alpha_1 \wedge \alpha_2$, then $\widehat{\alpha}^{\mathcal{A}}$ must be true iff $\widehat{\alpha}_1^{\mathcal{A}} = \widehat{\alpha}_2^{\mathcal{A}} = \text{true}$. The vocabulary of the formula $\text{CNF}(\text{NaiveGnd}_{\mathcal{A}}(\phi))$ is a Tseitin vocabulary for ϕ , and every model of that formula is a Tseitin structure for ϕ .

Definition 5 (Bound Structures). *Let ϕ be a formula, and \mathcal{A} be a structure for a sub-set of the vocabulary of ϕ . A bound structure for ϕ and \mathcal{A} is a partial Tseitin structure for ϕ that is a conservative strengthening of \mathcal{A} with respect to ϕ .*

Intuitively, a bound structure provides a way to represent the information from the instance together with additional information, including information about the Tseitin symbols in a grounding of ϕ , that we may derive (by any means), provided that information does not eliminate any intended models.

Let τ be the minimum vocabulary for bound structures for ϕ and \mathcal{A} . The bound structures for ϕ and \mathcal{A} with vocabulary τ form a lattice under the partial order \leq , with \mathcal{A} the minimum element. The maximum element is defined exactly for the atoms of $\text{CNF}(\text{NaiveGnd}_{\mathcal{A}}(\phi))$ which have the same truth value in every Tseitin τ -structure that satisfies ϕ . This is the structure produced by “Most Optimum Propagator” in [WMD10]).

Definition 6 (Grounding over a bound structure). Let $\hat{\mathcal{A}}$ be a bound structure for ϕ and \mathcal{A} . A formula ψ , over a Tseitin vocabulary for ϕ which includes $\hat{\mathcal{A}}$, is a grounding of ϕ over $\hat{\mathcal{A}}$ iff

1. if there is a total strengthening of $\hat{\mathcal{A}}$ that satisfies ϕ , then there is a one that satisfies ψ ;
2. if \mathcal{B} is a total Tseitin structure for ϕ which strengthens $\hat{\mathcal{A}}$, gives $\tilde{\mathcal{A}}$ the intended interpretation and satisfies ψ , then it satisfies ϕ .

A grounding ψ of ϕ over $\hat{\mathcal{A}}$ need not be a grounding of ϕ over \mathcal{A} . If we conjoin with ψ ground atoms representing the information contained in $\hat{\mathcal{A}}$, then we do obtain a grounding of ϕ over \mathcal{A} . In practice, we send just $\text{CNF}(\psi)$ to the SAT solver, and if a satisfying assignment is found, add the missing information back in at the time we construct a model for ϕ .

3.2 Top-down Grounding over a Bound Structure

Algorithm 2 produces a grounding of ϕ over a bound structure $\hat{\mathcal{A}}$ for \mathcal{A} . *Gnd* and *Simpl* are defined by mutual recursion. *Gnd* performs expansions and substitutions, while *Simpl* performs lookups in $\hat{\mathcal{A}}$ to see if the grounding of a sub-formula may be left out. *Eval* provides the base cases, evaluating ground atoms over $\sigma \cup \varepsilon \cup \tilde{\mathcal{A}} \cup \omega$ in $\hat{\mathcal{A}}$.

Algorithm 2 Top-Down Grounding over Bound Structure $\hat{\mathcal{A}}$ for ϕ and \mathcal{A}

$$\begin{aligned}
 \text{Gnd}_{\hat{\mathcal{A}}}(\phi, \theta) &= \begin{cases} \text{Eval}_{\hat{\mathcal{A}}}(P, \theta) & \phi \text{ is an atom } P(\bar{x}) \\ \neg \text{Eval}_{\hat{\mathcal{A}}}(P, \theta) & \phi \text{ is a negated atom } \neg P(\bar{x}) \\ \bigwedge_i \text{Simpl}_{\hat{\mathcal{A}}}(\psi_i, \theta) & \phi = \bigwedge_i \psi_i \\ \bigvee_i \text{Simpl}_{\hat{\mathcal{A}}}(\psi_i, \theta) & \phi = \bigvee_i \psi_i \\ \bigwedge_{a \in \mathcal{A}} \text{Simpl}_{\hat{\mathcal{A}}}(\psi, \theta \cup (x/\tilde{a})) & \phi = \forall x \psi \\ \bigvee_{a \in \mathcal{A}} \text{Simpl}_{\hat{\mathcal{A}}}(\psi, \theta \cup (x/\tilde{a})) & \phi = \exists x \psi \end{cases} \\
 \text{Eval}_{\hat{\mathcal{A}}}(P, \theta) &= \begin{cases} \top & \hat{\mathcal{A}} \models P[\theta] \\ \perp & \hat{\mathcal{A}} \models \neg P[\theta] \\ P(\bar{x})[\theta] & \text{o.w} \end{cases} \\
 \text{Simpl}_{\hat{\mathcal{A}}}(\psi, \theta) &= \begin{cases} \top & \hat{\mathcal{A}} \models [\psi][\theta] \\ \perp & \hat{\mathcal{A}} \models \neg [\psi][\theta] \\ \text{Gnd}_{\hat{\mathcal{A}}}(\psi, \theta) & \text{o.w} \end{cases}
 \end{aligned}$$

The stronger $\hat{\mathcal{A}}$ is, the smaller the ground formula produced by Algorithm 2. If we set $\hat{\mathcal{A}}$ to be undefined everywhere (i.e., to just give the domain), then Algorithm 2 produces $\text{NaiveGnd}_{\mathcal{A}}(\phi, \emptyset)$. If $\hat{\mathcal{A}}$ is set to \mathcal{A} , we get the reduced grounding obtained by evaluating instance symbols out of $\text{NaiveGnd}_{\mathcal{A}}(\phi)$.

Proposition 1. Algorithm 2 produces a grounding of ϕ over $\hat{\mathcal{A}}$.

3.3 Autarkies and Autark Subformulas

In the literature, an *autarky* [MS85] is informally a “self-sufficient” model for some clauses which does not affect the remaining clauses of the formula. An autark subformula is a subformula which is satisfied by an autarky. To see how an autark subformula may be produced during grounding, let $\lambda = \gamma_1 \vee \gamma_2$ and imagine that the value of subformula γ_1 is true according to our bound structure. Then λ will be true, regardless of the value of γ_2 , and the grounder will replace its subformula with its truth value, whereas in the case of naive grounding, the grounder does not have that information during the grounding. So it generates the set of clauses for this subformula as: $\{(\neg \lambda \vee \gamma_1 \vee \gamma_2), (\neg \gamma_1 \vee \lambda), (\neg \gamma_2 \vee \lambda)\}$. Now the propagation of the truth value of λ_1 and subsequently λ , results in elimination of all the three clauses, but the set of clauses

generated for γ_2 will remain in the CNF formula. We call γ_2 and the clauses made from that subformula autarkies.

The example suggests that this is a common phenomena and that the number of autarkies might be quite large in many groundings, as will be seen in Section 6.

4 Lifted Unit Propagation Structures

In this section we define $\mathcal{LUP}(\phi, \mathcal{A})$, and a method for constructing it.

Definition 7 ($\mathcal{LUP}(\phi, \mathcal{A})$). *Let Units denote the set of unit clauses that appears during the execution of UP on $\text{CNF}(\text{NaiveGnd}_{\mathcal{A}}(\phi))$. The LUP structure for ϕ and \mathcal{A} is the unique bound structure for ϕ and \mathcal{A} for which:*

$$\chi_{\lceil \psi \rceil}^{\mathcal{A}}(\bar{a}) = \begin{cases} \top & \lceil \psi \rceil(\bar{a}) \in \text{Units} \\ \perp & \neg \lceil \psi \rceil(\bar{a}) \in \text{Units} \\ \infty & \text{o.w} \end{cases} \quad (3)$$

Since Algorithm 2 produces a grounding, according to Definition 6, for any bound structure, it produces a grounding for ϕ over $\mathcal{LUP}(\phi, \mathcal{A})$.

To construct $\mathcal{LUP}(\phi, \mathcal{A})$, we use an inductive definition obtained from ϕ . In this inductive definition, we use distinct vocabulary symbols for the sets of tuples which $\hat{\mathcal{A}}$ sets to true and false. The algorithm works based on the notion of *True (False) bounds*:

Definition 8 (Formula-Bound). *A True (resp. False) bound for a subformula $\psi(\bar{x})$ according to bound structure $\hat{\mathcal{A}}$ is the relation denoted by T_{ψ} (resp. F_{ψ}) such that:*

1. $\bar{a} \in T_{\psi} \Leftrightarrow \lceil \psi \rceil^{\hat{\mathcal{A}}}(\bar{a}) = \top$
2. $\bar{a} \in F_{\psi} \Leftrightarrow \lceil \psi \rceil^{\hat{\mathcal{A}}}(\bar{a}) = \perp$

Naturally, when $\lceil \psi \rceil^{\hat{\mathcal{A}}}(\bar{a}) = \infty$, \bar{a} is not contained in either T_{ψ} or F_{ψ} .

The rules of the inductive definition are given in Table 1. These rules may be read as rules of FO(ID), the extension of classical logic with inductive definitions under the well-founded semantics [VGRS91,DT08], with free variables implicitly universally quantified. The *type* column indicates the type of the subformula, and the *rules* columns identify the rule for this subformula. Given a σ -structure \mathcal{A} , we may evaluate the definitions on \mathcal{A} , thus obtaining a set of concrete bounds for the subformulas of ϕ . The rules reflect the reasoning that UP can do. For example consider rule $(\forall_i \psi_i)$ of $\downarrow t$ for $\gamma(\bar{x}) = \psi_1(\bar{x}_1) \vee \dots \vee \psi_N(\bar{x}_N)$, and for some $i \in \{1, \dots, N\}$:

$$T_{\psi_i}(\bar{x}_i) \leftarrow T_{\gamma}(\bar{x}) \wedge \bigwedge_{j \neq i} F_{\psi_j}(\bar{x}_j).$$

This states that when a tuple \bar{a} satisfies γ but falsifies all disjuncts, ψ_j , of γ except for one, namely ψ_i , then it must satisfy ψ_i . As a starting point, we know the value of the instance predicates, and we also assume that ϕ is \mathcal{A} -satisfiable.

Example 2. Let $\phi = \forall x \neg I_1(x) \vee E_1(x)$, $\sigma = \{I_1, I_2\}$, and $\mathcal{A} = (\{1, 2, 3, 4\}; I_1^{\mathcal{A}} = \{1\})$. The relevant rules from Table (1) are:

$$\begin{aligned} T_{\neg I_1(x) \vee E_1(x)}(x) &\leftarrow T_{\phi} \\ T_{I_1}(x) &\leftarrow I_1(x) \\ F_{\neg I_1(x)}(x) &\leftarrow T_{I_1}(x) \\ T_{E_1(x)}(x) &\leftarrow T_{\neg I_1(x) \vee E_1(x)}(x) \wedge F_{\neg I_1(x)}(x) \\ T_{E_1(x)}(x) &\leftarrow T_{E_1(x)}(x) \end{aligned}$$

We find that $T_{E_1} = \{1\}$; in other words: $E_1(1)$ is true in each model of ϕ expanding \mathcal{A} .

Note that this inductive definition is monotone, because ϕ is in *Negation Normal Form (NNF)*.

type	$\downarrow t$ rules	type	$\uparrow t$ rules
$(\forall_i \psi_i)$	$T_{\psi_i}(\bar{x}_i) \leftarrow T_\gamma(\bar{x}) \wedge \bigwedge_{j \neq i} F_{\psi_j}(\bar{x}_j)$, for each i	$(\forall_i \psi_i)$	$T_\gamma(\bar{x}) \leftarrow \bigvee_i T_{\psi_i}(\bar{x}_i)$, for each i
$(\wedge_i \psi_i)$	$T_{\psi_i}(\bar{x}_i) \leftarrow T_\gamma(\bar{x})$, for each i	$(\wedge_i \psi_i)$	$T_\gamma(\bar{x}) \leftarrow \bigwedge_i T_{\psi_i}(\bar{x}_i)$, for each i
$\exists y \psi(\bar{x}, y)$	$T_\psi(\bar{x}, y) \leftarrow T_\gamma(\bar{x}) \wedge \forall y' \neq y \ F_\psi(\bar{x}, y')$	$\exists y \psi(\bar{x}, y)$	$T_\gamma(\bar{x}) \leftarrow \exists y T_\psi(\bar{x}, y)$
$\forall y \psi(\bar{x}, y)$	$T_\psi(\bar{x}, y) \leftarrow T_\gamma(\bar{x})$	$\forall y \psi(\bar{x}, y)$	$T_\gamma(\bar{x}) \leftarrow \forall y T_\psi(\bar{x}, y)$
$P(\bar{x})$	$T_P(\bar{x}) \leftarrow T_\gamma(\bar{x})$	$P(\bar{x})$	$T_\gamma(\bar{x}) \leftarrow T_P(\bar{x})$
$\neg P(\bar{x})$	$F_P(\bar{x}) \leftarrow T_\gamma(\bar{x})$	$\neg P(\bar{x})$	$T_\gamma(\bar{x}) \leftarrow F_P(\bar{x})$

type	$\downarrow f$ rules	type	$\uparrow f$ rules
$(\forall_i \psi_i)$	$F_{\psi_i}(\bar{x}_i) \leftarrow F_\gamma(\bar{x})$, for each i	$(\forall_i \psi_i)$	$F_\gamma(\bar{x}) \leftarrow \bigwedge_i F_{\psi_i}(\bar{x}_i)$, for each i
$(\wedge_i \psi_i)$	$F_{\psi_i}(\bar{x}_i) \leftarrow F_\gamma(\bar{x}) \wedge \bigwedge_{j \neq i} T_{\psi_j}(\bar{x}_j)$, for each i	$(\wedge_i \psi_i)$	$F_\gamma(\bar{x}) \leftarrow \bigvee_i F_{\psi_i}(\bar{x}_i)$, for each i
$\exists y \psi(\bar{x}, y)$	$F_\psi(\bar{x}, y) \leftarrow F_\gamma(\bar{x})$	$\exists y \psi(\bar{x}, y)$	$F_\gamma(\bar{x}) \leftarrow \forall y F_\psi(\bar{x}, y)$
$\forall y \psi(\bar{x}, y)$	$F_\psi(\bar{x}, y) \leftarrow F_\gamma(\bar{x}) \wedge \forall y' \neq y \ T_\psi(\bar{x}, y')$	$\forall y \psi(\bar{x}, y)$	$F_\gamma(\bar{x}) \leftarrow \exists y F_\psi(\bar{x}, y)$
$P(\bar{x})$	$F_P(\bar{x}) \leftarrow F_\gamma(\bar{x})$	$P(\bar{x})$	$F_\gamma(\bar{x}) \leftarrow F_P(\bar{x})$
$\neg P(\bar{x})$	$T_P(\bar{x}) \leftarrow F_\gamma(\bar{x})$	$\neg P(\bar{x})$	$F_\gamma(\bar{x}) \leftarrow T_P(\bar{x})$

Table 1: Rules for Bounds Computation

4.1 LUP Structure Computation

Our method for constructing $\mathcal{LUP}(\phi, \mathcal{A})$ is given in Algorithm 3. Several lines in the algorithm require explanation. In line 1, the $\downarrow f$ rules are omitted from the set of constructed rules. Because ϕ is in NNF, the $\downarrow f$ rules do not contribute any information to the set of bounds. To see this, observe that every $\downarrow f$ rule has an atom of the form $F_\gamma(\bar{x})$ in its body. Intuitively, for one of these rules to contribute a defined bound, certain information must have previously been obtained regarding bounds for its parent. It can be shown, by induction, that, in every case, the information about a bound inferred by an application of a $\downarrow f$ rule must have previously been inferred by a $\uparrow f$ rule. In line 2 of the algorithm we compute bounds using only the two sets of rules, $\downarrow t$ and $\uparrow f$. This is justified by the fact that applying $\{\uparrow t, \downarrow t, \uparrow f\}$ to a fixpoint has the same effect as applying $\{\downarrow t, \uparrow f\}$ to a fixpoint and then applying the $\uparrow t$ rules afterwards. So we postpone the execution of the $\uparrow t$ rules to line 7.

Line 3 checks for the case that the definition has no model, which is to say that the rules allow us to derive that some atom is both in the true bound and the false bound for some subformula. This happens exactly when UP applied to the naive grounding would detect inconsistency.

Finally, in lines 6 and 7 we throw away the true bounds for all non-atomic subformulas, and then compute new bounds by evaluating the $\uparrow t$ rules, taking already computed bounds (with true bounds for non-atoms set to empty) as the initial bounds in the computation. To see why, observe that the true bounds computed in line 2 are based on the assumption that ϕ is \mathcal{A} -satisfiable. So $\lceil \phi \rceil$ is set to true which stops the top-down bounded grounding algorithm of Section 3.2 from producing a grounding for ϕ . That is because the *Simpl* function, considering the true bound for the ϕ , simply returns \top instead of calling $Gnd_{\hat{\mathcal{A}}}(\cdot, \cdot)$ on subformulas of the ϕ . This also holds for all the formulas with true-bounds, calculated this way, except for the atomic formulas. So, we delete these true bounds based on the initial unjustified assumption, and

Algorithm 3 Computation of $\mathcal{LUP}(\phi, \mathcal{A})$

- 1: Construct the rules $\{\uparrow t, \downarrow t, \uparrow f\}$
 - 2: Compute bounds by evaluating the inductive definition $\{\downarrow t, \uparrow f\}$
 - 3: **if** Bounds are inconsistent **then**
 - 4: **return** “ \mathcal{A} has no solution”
 - 5: **end if**
 - 6: Throw away $T_\psi(\bar{x})$ for all non-atomic subformulas $\psi(\bar{x})$
 - 7: Compute new bounds by evaluating the inductive definition $\{\uparrow t\}$
 - 8: **return** LUP structure constructed from the computed bounds, according to Definition 8.
-

then construct the correct true bounds by application of the $\uparrow t$ rules, in line 7. This is the main reason for postponing the execution of $\uparrow t$ rules.

5 Bottom-up Grounding over Bound Structures

The grounding algorithm we use in Enfragmo constructs a grounding by a bottom-up process that parallels database query evaluation, based on an extension of the relational algebra. We give a rough sketch of the method here: further details can be found in, e.g., [Moh04,PLTG07]. Given a structure (database) \mathcal{A} , a boolean query is a formula ϕ over the vocabulary of \mathcal{A} , and query answering is evaluating whether ϕ is true, i.e., $\mathcal{A} \models \phi$. In the context of grounding, ϕ has some additional vocabulary beyond that of \mathcal{A} , and producing a reduced grounding involves evaluating out the instance vocabulary, and producing a ground formula representing the expansions of \mathcal{A} for which ϕ is true.

For each sub-formula $\alpha(\bar{x})$ with free variables \bar{x} , we call the set of reduced groundings for α under all possible ground instantiations of \bar{x} an answer to $\alpha(\bar{x})$. We represent answers with tables on which the extended algebra operates. An X-relation, in databases, is a k -ary relation associated with a k -tuple of variables X, representing a set of instantiations of the variables of X. Our grounding method uses extended X-relations, in which each tuple \bar{a} is associated with a formula. In particular, if R is the answer to $\alpha(\bar{x})$, then R consists of the pairs $(\bar{a}, \alpha(\bar{a}))$. Since a sentence has no free variables, the answer to a sentence ϕ is a zero-ary extended X-relation, containing a single pair $(\langle \rangle, \psi)$, associating the empty tuple with formula ψ , which is a reduced grounding of ϕ .

The relational algebra has operations corresponding to each connective and quantifier in FO: complement (negation); join (conjunction); union (disjunction), projection (existential quantification); division or quotient (universal quantification). Each generalizes to extended X-relations. If $(\bar{a}, \alpha(\bar{a})) \in \mathcal{R}$ then we write $\delta_{\mathcal{R}}(\bar{a}) = \alpha(\bar{a})$. For example, the join of extended X-relation \mathcal{R} and extended Y-relation \mathcal{S} (both over domain A), denoted $\mathcal{R} \bowtie \mathcal{S}$, is the extended $X \cup Y$ -relation $\{(\bar{a}, \psi) \mid \bar{a} : X \cup Y \rightarrow A, \bar{a}|_X \in \mathcal{R}, \bar{a}|_Y \in \mathcal{S}, \text{ and } \psi = \delta_{\mathcal{R}}(\bar{a}|_X) \wedge \delta_{\mathcal{S}}(\bar{a}|_Y)\}$; It is easy to show that, if \mathcal{R} is an answer to $\alpha_1(\bar{x})$ and \mathcal{S} is an answer to $\alpha_2(\bar{y})$ (both wrt \mathcal{A}), then $\mathcal{R} \bowtie \mathcal{S}$ is an answer to $\alpha_1(\bar{x}) \wedge \alpha_2(\bar{y})$. The analogous property holds for the other operators.

To ground with this algebra, we define the answer to atomic formula $P(\bar{x})$ as follows. If P is an instance predicate, the answer is the set of tuples (\bar{a}, \top) , for $\bar{a} \in P^{\mathcal{A}}$. If P is an expansion predicate, the answer is the set of all tuples $(\bar{a}, P(\bar{a}))$, for \bar{a} a tuple of elements from the domain of \mathcal{A} . Then we apply the algebra inductively, bottom-up, on the structure of the formula. At the top, we obtain the answer to ϕ , which is a relation containing only the pair $(\langle \rangle, \psi)$, where ψ is a reduced grounding of ϕ wrt \mathcal{A} .

Example 3. Let $\sigma = \{P\}$ and $\varepsilon = \{E\}$, and let \mathcal{A} be a σ -structure with $P^{\mathcal{A}} = \{(1, 2, 3), (3, 4, 5)\}$. The following extended relation \mathcal{R} is an answer to $\phi_1 \equiv P(x, y, z) \wedge E(x, y) \wedge E(y, z)$:

x	y	z	ψ
1	2	3	$E(1, 2) \wedge E(2, 3)$
3	4	5	$E(3, 4) \wedge E(4, 5)$

Observe that $\delta_{\mathcal{R}}(1, 2, 3) = E(1, 2) \wedge E(2, 3)$ is a reduced grounding of $\phi_1[(1, 2, 3)] = P(1, 2, 3) \wedge E(1, 2) \wedge E(2, 3)$, and $\delta_{\mathcal{R}}(1, 1, 1) = \perp$ is a reduced grounding of $\phi_1[(1, 1, 1)]$.

The following extended relation is an answer to $\phi_2 \equiv \exists z \phi_1$:

x	y	ψ
1	2	$E(1, 2) \wedge E(2, 3)$
3	4	$E(3, 4) \wedge E(4, 5)$

Here, $E(1, 2) \wedge E(2, 3)$ is a reduced grounding of $\phi_2[(1, 2)]$. Finally, the following represents an answer to $\phi_3 \equiv \exists x \exists y \phi_2$, where the single formula is a reduced grounding of ϕ_3 .

ψ
$[E(1, 2) \wedge E(2, 3)] \vee [E(3, 4) \wedge E(4, 5)]$

To modify the algorithm to ground using $\mathcal{LUP}(\phi, \mathcal{A})$ we need only change the base case for expansion predicates. To be precise, if P is an expansion predicate we set the answer to $P(\bar{x})$ to the set of pairs (\bar{a}, ψ) such that:

$$\psi = \begin{cases} P(\bar{a}) & \text{if } P^{\mathcal{LUP}(\phi, \mathcal{A})}(\bar{a}) = \infty \\ \top & \text{if } P^{\mathcal{LUP}(\phi, \mathcal{A})}(\bar{a}) = \top \\ \perp & \text{if } P^{\mathcal{LUP}(\phi, \mathcal{A})}(\bar{a}) = \perp. \end{cases}$$

Observe that bottom-up grounding mimics the second phase of Algorithm 3, i.e., a bottom-up truth propagation, except that it also propagates the falses. So, for bottom up grounding, we can omit line 7 from Algorithm 3.

Proposition 2. *Let $(\langle \rangle, \psi)$ be the answer to sentence ϕ wrt \mathcal{A} after LUP initialization, then:*

$$Gnd_{\mathcal{LUP}(\phi, \mathcal{A})}(\phi, \emptyset) \equiv \psi$$

where $Gnd_{\mathcal{LUP}(\phi, \mathcal{A})}(\phi, \emptyset)$ is the result of top-down grounding Algorithm 2 of ϕ over LUP structure $\mathcal{LUP}(\phi, \mathcal{A})$.

This bottom-up method uses only the reduct of $\mathcal{LUP}(\phi, \mathcal{A})$ defined by $\sigma \cup \varepsilon \cup \tilde{A}$, not the entire LUP structure.

6 Experimental Evaluation of LUP

In this section we present an empirical study of the effect of LUP on grounding size and on grounding and solving times. We also compare LUP with GWB in terms of these same measures. The implementation of LUP is within our bottom-up grounder Enfragmo, as described in this paper, and the implementation of GWB is in the top-down grounder GIDL, which is described in [WMD08b, WMD10]. GIDL has several parameters to control the precision of the bounds computation. In our experiments we use the default settings. We used MINISAT as the ground solver for Enfragmo. GIDL produces an output specifically for the ground solver MINISAT(ID), and together they form the IDP system [WMD08d].

We report data for instances of three problems: Latin Square Completion, Bounded Spanning Tree and Sudoku. The instances are latin_square.17068* instances of Normal Latin Square Completion, the 104_rand.45_250.* and 104_rand.35_250.* instances of BST, and the ASP contest 2009 instances of Sudoku from the Asparagus repository³. All experiments were run on a Dell Precision T3400 computer with a quad-core 2.66GHz Intel Core 2 processor having 4MB cache and 8GB of RAM, running CentOS 5.5 with Linux kernel 2.6.18.

In Tables 2 and 4, columns headed “Literals” or “Clauses” give the number of literals or clauses in the CNF formula produced by Enfragmo without LUP (our baseline), or these values for other grounding methods expressed as a percentage of the baseline value. In Tables 3 and 5, all values are times seconds. All values give are means for the entire collection of instances. Variances are not given, because they are very small. We split the instances of BST, into two sets, based on the number of nodes (35 or 45), because these two groups exhibit somewhat different behaviour, but within the groups variances are also small. In all tables, the minimum (best) values for each row are in bold face type, to highlight the conditions which gave best performance.

Table 2 compares the sizes of CNF formulas produced by Enfragmo without LUP (the base line) with the formulas obtained by running UP on the baseline formulas and by running Enfragmo with LUP. Clearly LUP reduces the size at least as much as UP, and usually reduces the size much more, due to the removal of autarkies.

Total time for solving a problem instance is composed of grounding time and SAT solving time. Table 3 compares the grounding and SAT solving time with and without LUP bounds. It is evident that the SAT solving time is always reduced with LUP. This reduction is due to the elimination of the unit clauses and autark subformulas from the grounding. Autark subformula elimination also affects the time required to convert the ground formula to CNF which reduces the grounding time, but in some cases the overhead

³ <http://asparagus.cs.uni-potsdam.de>

	Enfragmo		Enfragmo+UP (%)		Enfragmo+LUP (%)	
Problem	Literals	Clauses	Literals	Clauses	Literals	Clauses
Latin Square	7452400	2514100	0.07	0.07	0.07	0.07
BST 45	22924989	9061818	0.96	0.96	0.24	0.24
BST 35	8662215	3415697	0.95	0.96	0.37	0.37
Sudoku	2875122	981668	0.17	0.18	0.07	0.08

Table 2: Impact of LUP on the size of the grounding. The first two columns give the numbers of literals and clauses in groundings produced by Enfragmo without LUP (the baseline). The other columns give these measures for formulas produced by executing UP on the baseline groundings (Enfragmo+UP), and for groundings produced by Enfragmo with LUP (Enfragmo+LUP), expressed as a fraction baseline values.

	Enfragmo			Enfragmo with LUP			Speed Up Factor		
Problem	Gnd	Solving	Total	Gnd	Solving	Total	Gnd	Solving	Total
Latin Square	0.89	1.39	2.28	3.27	0.34	3.61	-2.38	1.05	-1.33
BST 45	6.08	7.56	13.64	2	1.74	3.74	4.07	5.82	9.9
BST 35	2.13	2.14	4.27	1.07	0.46	1.53	1.06	1.68	2.74
Sudoku	0.46	1.12	1.59	2.08	0.26	2.34	-1.62	0.86	-0.76

Table 3: Impact of LUP on reduction in both grounding and (SAT) solving time. Grounding time here includes LUP computations and CNF generation.

imposed by LUP computation may not be made up for by this reduction. As the table shows, when LUP outperforms the normal grounding we get factor of 3 speed-ups, whereas when it loses to normal grounding the slowdown is by a factor of 1.5.

Table 4 compares the size reductions obtained by LUP and by GWB in GIDL. The output of GIDL contains clauses and rules. The rules are transformed to clauses in (MINISAT(ID)). The measures reported here are after that transformation. LUP reduces the size much more than GWB, in most of the cases. This stems from the fact that GIDL’s bound computation does not aim for completeness wrt unit propagation. This also affects the solving time because the CNF formulas are much smaller with LUP as shown in Table 5. Table 5 shows that Enfragmo with LUP and MINISAT is always faster than GIDL with MINISAT(ID) with or without bounds, and it is in some cases faster than Enfragmo without LUP.

7 Discussion

In the context of grounding-based problem solving, we have described a method we call lifted unit propagation (LUP) for carrying out a process essentially equivalent to unit propagation before and during grounding. Our experiments indicate that the method can substantially reduce grounding size – even more than unit propagation itself, and sometimes reduce total solving time as well.

	Enfragmo (no LUP)		GIDL (no bounds)		Enfragmo with LUP		GIDL with bounds	
Problem	Literals	Clauses	Literals	Clauses	Literals	Clauses	Literals	Clauses
Latin Square	7452400	2514100	0.74	0.84	0.07	0.07	0.59	0.61
BST 45	22924989	9061818	0.99	1.02	0.24	0.24	0.25	0.24
BST 35	8662215	3415697	1.01	1.04	0.37	0.37	0.39	0.39
Sudoku	2875122	981668	0.56	0.6	0.07	0.08	0.38	0.39

Table 4: Comparison between the effectiveness of LUP and GIDL Bounds on reduction in grounding size. The columns under Enfragmo show the actual grounding size whereas the other columns show the ratio of the grounding size relative to that of Enfragmo (without LUP).

Problem	Enfragmo			IDP			Enfragmo+LUP			IDP (Bounds)		
	Gnd	Solving	Total	Gnd	Solving	Total	Gnd	Solving	Total	Gnd	Solving	Total
Latin Square	0.89	1.39	2.28	3	4.63	7.63	3.27	0.34	3.61	2.4	3.81	6.21
BST 45	6.08	7.56	13.64	7.25	20.84	28.09	2	1.74	3.74	1.14	4.45	5.59
BST 35	2.13	2.14	4.27	2.63	6.31	8.94	1.07	0.46	1.53	0.67	2.73	3.4
Sudoku	0.46	1.12	1.59	1.81	1.3	3.11	2.08	0.26	2.34	2.85	0.51	2.37

Table 5: Comparison of solving time for Enfragmo and IDP, with and without LUP/bounds.

Our work was motivated by the results of [WMD08b,WMD10], which presented the method we have referred to as *GWB*. In *GWB*, bounds on sub-formulas of the specification formula are computed without reference to an instance structure, and represented with FO formulas. The grounding algorithm evaluates instantiations of these bound formulas on the instance structure to determine that certain parts of the naive grounding may be left out. If the bound formulas exactly represent the information unit propagation can derive, then LUP and *GWB* are equivalent (though implemented differently). However, generally the *GWB* bounds are weaker than the LUP bounds, for two reasons. First, they must be weaker, because no FO formula can define the bounds obtainable with respect to an arbitrary instance structure. Second, to make the implementation in *GIDL* efficient, the computation of the bounds is heuristically truncated. This led us to ask how much additional reduction in formula size might be obtained by the complete LUP method, and whether the LUP computation could be done fast enough for this extra reduction to be useful in practice.

Our experiments with the Enfragmo and *GIDL* grounders show that, at least for some kinds of problems and instances, using LUP can produce much smaller groundings than the *GWB* implementation in *GIDL*. In our experiments, the total solving times for Enfragmo with ground solver MINISAT were always less than those of *GIDL* with ground solver MINISAT(ID). However, LUP reduced total solving time of Enfragmo with MINISAT significantly in some cases, and increased it — albeit less significantly — in others. Since there are many possible improvements of the LUP implementation, the question of whether LUP can be implemented efficiently enough to be used all the time remains unanswered.

Investigating more efficient ways to do LUP, such as by using better data structures, is a subject for future work, as is consideration of other approximate methods such, as placing a heuristic time-out on the LUP structure computation, or dovetailing of the LUP computation with grounding. We also observed that the much of the reduction in grounding size obtained by LUP is due to identification of autark sub-formulas. These cannot be eliminated from the naive grounding by unit propagation. Further investigation of the importance of these in practice is another direction we are pursuing. One more direction we are pursuing is the study of methods for deriving even stronger information that represented by the LUP structure, to further reduce ground formula size, and possibly grounding time as well.

Acknowledgements

The authors are grateful to Marc Denecker, Johan Wittocx, Bill MacReady, Calvin Tang, Amir Aavani, Shahab Tasharoffi, Newman Wu, D-Wave Systems, MITACS, and NSERC.

References

- ATÜ⁺10. Amir Aavani, Shahab Tasharoffi, Gulay Ünel, Eugenia Ternovska, and David G. Mitchell. Speed-up techniques for negation in grounding. In *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 13–26. Springer, 2010.
- AWTM11. Amir Aavani, Xiongnan (Newman) Wu, Eugenia Ternovska, and David G. Mitchell. Grounding formulas with complex terms. In *Canadian AI*, volume 6657 of *LNCS*, pages 13–25. Springer, 2011.
- DT08. Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM Trans. Comput. Logic*, 9:14:1–14:52, April 2008.
- DW08. M. Denecker and J. Wittocx. Personal communication, 2008.
- ET06. D. East and M. Truszczyński. Predicate-calculus based logics for modeling and solving search problems. *ACM Trans. Comput. Logic (TOCL)*, 7(1):38 – 83, 2006.

- GKK⁺08. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to gringo, clasp, clingo, and iclingo. Unpublished draft, 2008.
- Kle52. S.C. Kleene. *Introduction to Metamathematics*. Bibliotheca Mathematica. North-Holland, 1952.
- LPF⁺06. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dl_v system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.
- Moh04. Raheleh Mohebbi. A method for solving NP search based on model expansion and grounding. Master's thesis, Simon Fraser University, 2004.
- MS85. B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Appl. Math.*, 10:287–295, March 1985.
- MT11. David Mitchell and Eugenia Ternovska. Knowledge representation, search problems and model expansion. In *Knowing, Reasoning and Acting: Essays in Honour of Hector J. Levesque*, pages 347–362, 2011.
- MTHM06. David Mitchell, Eugenia Ternovska, Faraz Hach, and Raheleh Mohebbi. Model expansion as a framework for modelling and solving search problems. Technical Report TR 2006-24, School of Computing Science, Simon Fraser University, December 2006.
- PLTG07. M. Patterson, Y. Liu, E. Ternovska, and A. Gupta. Grounding for model expansion in k-guarded formulas with inductive definitions. In *Proc. IJCAI'07*, 2007.
- TJ07. Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In *TACAS*, volume 4424 of *LNCS*, pages 632–647. Springer, 2007.
- Tse68. G.S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115 – 125, 1968.
- VGRS91. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38:619–649, July 1991.
- WMD08a. Johan Wittocx, Maarten Mariën, and Marc Denecker. GidL: A grounder for FO+. In *In Proc., Twelfth International Workshop on NMR*, pages 189–198, September 2008.
- WMD08b. Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding with bounds. In *AAAI*, pages 572–577, 2008.
- WMD08c. Johan Wittocx, Maarten Mariën, and Marc Denecker. The IDP system: a model expansion system for an extension of classical logic. In *Proc., LaSh 2008*, 2008.
- WMD08d. Johan Wittocx, Maarten Mariën, and Marc Denecker. The IDP system: a model expansion system for an extension of classical logic. In Marc Denecker, editor, *LaSh*, pages 153–165, 2008.
- WMD10. Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *J. Artif. Intell. Res. (JAIR)*, 38:223–269, 2010.